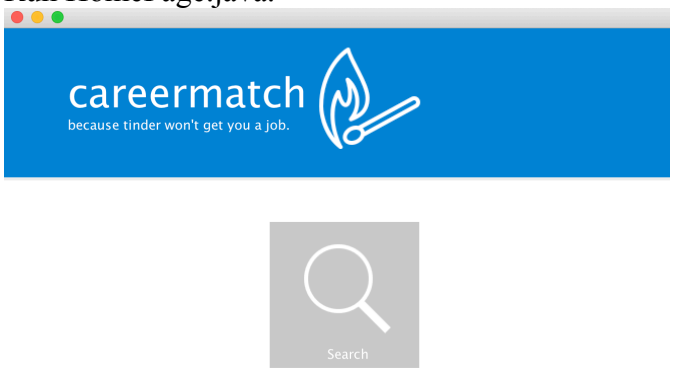
User manual: careermatch

SETTING UP THE PROJECT IN YOUR IDE:

1) Import the project into your IDE like you normally would.
2) Add all the .jars in the jars folder provided to your project.

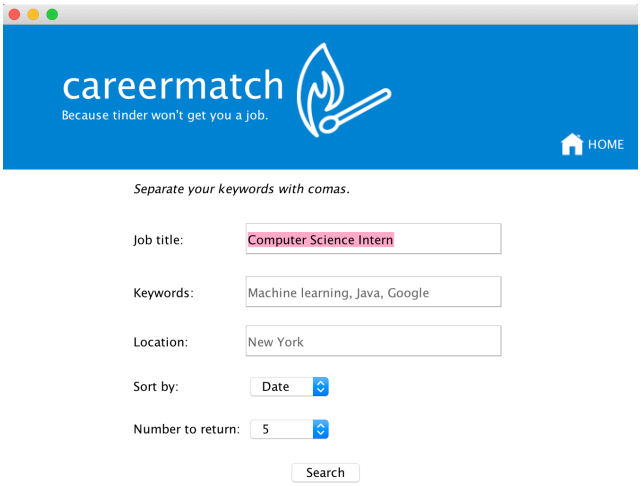
RUNNING THE PROJECT:

1) Run HomePage.java.

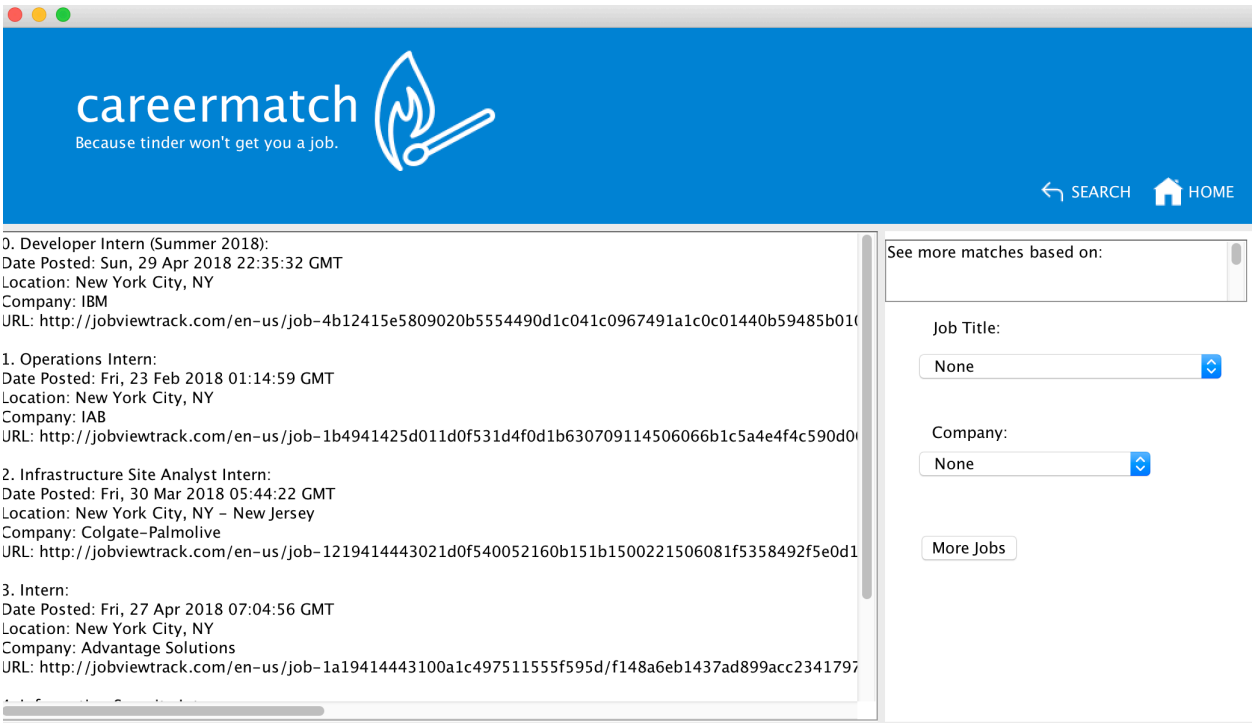


2) Click on Search (the big gray square). Note that clicking on Java Swing is strange so it may take a couple of clicks (or harder clicks) for the change to happen.

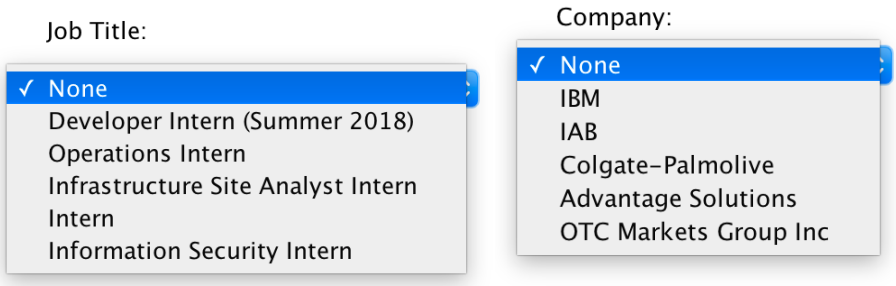
3) Input your search query. The inputs you see are the default inputs that serve as example. For job title, simply input a job title. For keywords, input things like skills required and company. For location, input a state, country, continent, or "world" if you want a worldwide search result thread. Select what you want the scraper to scrape by (date vs. salary), and input the number of jobs that you want to be displayed. Click "Search" when you have finished inputting your query. You know when the program is running when your mouse switches to a waiting mouse and the top of the window says "Loading…". This search will take a couple of minutes as the engine is going to the API and scrape for jobs and their descriptions (so the speed will depend on your internet speed).



4) A couple of things can happen when you click "search". If your inputs are unlikely (astronaut with bacon as keyword in Missouri), you may receive a pop up screen telling you that there are no matches found, in which case you can close that screen and input a different set of criteria. Else you will receive a screen that display all your results:



5) Now you have an option to display even more results based on either/or job title and company. The job titles will be all the jobs that are on your current matched list that you see, and the companies are the places that those jobs belong to.



You can either select a job title, a company, or both to see more search results. However, if you leave both blanks (if you chose "None" for both), then an error screen will pop up and prompt you to select at least one.

6) Once you are done selecting your criteria, click on "More jobs" to see more matches. Please wait a couple of minutes for the scraper to retrieve necessary information and for the engine to calculate results. You know when search is in progress when the mouse is a loading mouse:

On Windows:

On Mac:

7) Once the search is completed, you should see the results popped up on a different window:

8) At any point in the program (unless it's in searching progress) you can exit out of the program by clicking to red x on the top panel of the frame (just like a normal window).

THE IDEAS BEHIND THE ENGINE:

0) Retrieving the information
   We utilized the CareerJet API to scrape some of the information that would be used for our search ranking and triadic closure functionality. However, we were unable to access the full job description from what the API returned, so we had to create a headless browser using HtmlUnit to access dynamically loaded data. We then were able to scrape the full description, and we added all this information for each post into a JobPost object.

1) Search ranking based on tf-idf
   Similar to what we did in class with document rankings on HW-4, the engine uses the Vector-Space model to rank jobs based on their cosine similarity values when compared to the user's query. The criteria that we used to determine cosine similarity are job title, job description, and location, all inputted by user.

2) Search ranking based on triadic closure
   As we learned in class, we wanted to be able to recommend jobs to our users when a user decided that s/he liked a job. So, we wanted to create a notion of a strongly connected edges between job-posts so that when they created a strongly connected edge between the job post and user, that we could use the idea of triadic closure to connect users to job postings that are similar. We did this by looking at jobs which either were at the same company or contained some of the same words within the job title in which we had a threshold of similarity before the program considered two job postings to be strongly connected.

CHALLENGES AND CONCLUSIONS:

Along the way, we encountered some challenges in terms of retrieving the information from CareerJet and processing it. There were some errors such as 404 Not Found that we had to process since some jobs may have been removed from the database. We also found that some jobs could not have their descriptions displayed. There are some limitations for our engine in terms of user inputs. We did not process typos or unusual inputs, which could have caused (though minor) changes in the results outputted. Another challenge was that our IP addresses got blocked from the system because of continuous requests within a short span of time.

Our engine's efficiency is limited by the information retrieval since the engine has to go online and scrape jobs from CareerJet's API, which takes time relative to the speed of the internet connection on the user's end. One way that we could improve this is to directly store/maintain our own database of jobs (which would be very big and somewhat defeat the purpose of relying on the CareerJet API).

If time allowed, we would have added additional features search as saving jobs and triadic closure ranking based on a wider range of criteria (industry, contract type, etc.). We could also allow users to upload their resumes (as query) and recommend jobs that best match their resumes.